Security Protocol Specification and Verification with *AnBx*

Michele Bugliesi[a], Stefano Calzavara[a], Sebastian Mödersheim[b], Paolo Modesti[c,1]

[a] *Dipartimento di Scienze Ambientali Informatica e Statistica, Università Ca' Foscari Venezia, Venezia, Italy*
[b] *DTU Compute, Danmarks Tekniske Universitet, Kgs. Lyngby, Denmark*
[c] *School of Computing Science, Newcastle University, Newcastle upon Tyne, United Kingdom*

Abstract

Designing distributed protocols is complex and requires actions at very different levels: from the design of an interaction flow supporting the desired application-specific guarantees, to the selection of the most appropriate network-level protection mechanisms. To tame this complexity, we propose *AnBx*, a formal protocol specification language based on the popular *Alice & Bob* notation. *AnBx* offers channels as the main abstraction for communication, providing different authenticity and/or confidentiality guarantees for message transmission. *AnBx* extends existing proposals in the literature with a novel notion of *forwarding* channels, enforcing specific security guarantees from the message originator to the final recipient along a number of intermediate forwarding agents. We give a formal semantics of *AnBx* in terms of a state transition system expressed in the AVISPA Intermediate Format. We devise an ideal channel model and a possible cryptographic implementation, and we show that, under mild restrictions, the two representations coincide, thus making *AnBx* amenable to automated verification with different tools. We demonstrate the benefits of the declarative specification style distinctive of *AnBx* by revisiting the design of two existing e-payment protocols, *iKP* and *SET*.

Keywords: Protocol specification, protocol verification, model-checking, e-payment

1. Introduction

The *Alice & Bob* notation, also known as *protocol narrations*, is a popular device which has been widely adopted in the literature as the basis of several security protocol specification frameworks [1, 2, 3, 4, 5, 6]. In such frameworks, the semantics of the specification languages is defined by a translation into lower level formats, amenable to model-checking and automated verification. Besides making verification possible, the translation semantics provides for a clean separation between the abstract specification of the protocol structure and the details of its implementation, which may be generated directly from the specification [6, 7, 8, 9, 10, 11, 12]. This separation has a beneficial impact on both the specification and the implementation: on the one hand, it helps focusing on application-level properties, staying away from unnecessary low-level details; on the other hand, it contributes to strengthening the implementation and to ensure the protocol end-to-end security, by delegating to the compiler the selection of the most adequate core implementation components.

Channel abstractions make a further step in the same direction: they help designing distributed applications irrespective of the cryptographic mechanisms needed to protect communication, by interpreting channels as a secure communication medium with built-in protection against certain attacks (e.g., on confidentiality).

How these properties are actually ensured represents a different design aspect, which might not be a concern of the application designer at all, and may be left to the compiler.

Related work. Several papers in the literature have taken this approach, and developed it along different directions. First, there are papers that

Email addresses: bugliesi@unive.it (Michele Bugliesi), calzavara@dais.unive.it (Stefano Calzavara), samo@imm.dtu.dk (Sebastian Mödersheim), paolo.modesti@sunderland.ac.uk (Paolo Modesti)

[1] Current address: *Department of Computing, Engineering and Technology, University of Sunderland, Sunderland, United Kingdom*

propose the definition and implementation of different channel abstractions, based on cryptographic realizations and interaction patterns. Abadi et al. propose a process calculus with native constructs for authentication and discuss a possible cryptographic implementation [13]. Adão and Fournet design a variant of the pi-calculus with secure communication and describe its computationally sound compilation into a concrete implementation [14]. Other authors explore the idea of compiling secure protocols for distributed sessions from convenient ML abstractions based on session types, a powerful formalism used to structure interaction and reason over communicating processes and their behaviour [15, 16].

Another line of research, instead, is more focused on reasoning about channels and their ideal behaviour in an abstract way. Dilloway and Lowe present a hierarchy of secure channels and discuss their relative strengths [17]. Bugliesi and Focardi devise secure channel abstractions in a process algebraic setting and reason about the relative power of a low-level adversary [18]. Armando et al. model different channel types using set-rewriting and linear temporal logic [19]. Kamil and Lowe adapt the Strand Spaces model to deal with secure channels, providing different security guarantees [20, 21].

Mödersheim and Viganò consider both an abstract characterization and a concrete realization of channels, showing that both characterizations coincide; the paper defines also the notion of channels as goals and proves a related compositionality result [22]. The same authors also formalize some easy-to-check static conditions that support a large class of channels and applications and that are sufficient for vertical security protocol composition [23]. These works also demonstrated that Alice and Bob notation is ideal for the combination with the channel notation, and channel types were integrated both in the languages AnB [4] and SPS [6]. In these papers, the focus is on giving a very general and concise semantics to Alice and Bob notation, namely defining with a few mathematically simple principles the semantics in presence of an arbitrary algebraic theory. With respect to this semantics, [6] proves the correctness of a translator to formal models and implementations. Our paper is based on this semantic machinery for the cryptographic handling of messages, and defines a rich set of channels on top of this basis.

We should mention two more related works on channels. Gibson-Robinson employs the notion of channel (and their properties) for the analysis of multi-layer security protocols [24]. Finally, Sprenger and Basin consider a refinement approach where cryptographic protocols are synthesised from high-level security goals; one of the steps of the refinement process builds on the usage of channel abstractions [25, 26].

Contributions. In the present paper we develop channels one step further, generalizing them to capture the notion of *forwarding channel*, a critical abstraction for designing and reasoning about complex protocols involving three or more communicating parties. A typical scenario for such protocols is represented by e-commerce transactions, in which a customer requires a merchant to certify that her payment has been cleared out, and the merchant provides that evidence by forwarding to the customer the notification she received from the credit card issuer. Similarly, single sign-on protocols usually involve an authenticity-preserving forwarding of access tokens from a trusted third-party to different clients. This kind of interactions may be modelled by session types, since they are typically developed on top of very expressive calculi and languages, but it is not accounted for in existing protocol narration frameworks with channel abstractions. Including forwarding in these frameworks is important, given their wide popularity and ease of use.

We develop the novel concept of forwarding channel as part of *AnBx*, a formal specification language that we introduce by conservatively extending the semantics of the *AnB* language [4]. *AnBx* includes modes for all kinds of message forwarding, where all or some of the properties of the original transmission are preserved upon relaying. In our characterization, we provide both an abstract interpretation of channels that captures their ideal behavior, and a cryptographic implementation, and we prove a formal equivalence between the two characterizations. Both interpretations are based on a translation to the AVISPA Intermediate Format, hence *AnBx* is directly available for automated verification with the different tools that use this format, such as OFMC [27].

We demonstrate the practical effectiveness of our approach by an analysis and re-engineering of two real-life e-payment protocols: *iKP* (Internet Keyed Payment [28, 29]), and *SET* (Secure Electronic Transaction [30, 31, 32]). Though

both protocols could be expressed in their full complexity in *AnBx*, we rely on the abstract channels available in the language to factor out the cryptographic aspects almost entirely. The resulting protocols are more concise, easier to understand and, interestingly, more efficient to verify than the original versions.

In addition, the *AnBx* formulations strengthen the original specifications, in that they enjoy stronger security goals and properties. As a byproduct of our comparative analysis, we also find a (to the best of our knowledge) new flaw in the original specification of *iKP*, and propose an amended version that rectifies the problem.

Moreover, the Java implementation of the revised versions of *iKP* and *SET* proved to behave well at run-time [11, 12], in some cases executing even faster than their original counterparts. This demonstrates that the benefits of using a language like *AnBx* are not limited to the design and verification levels but they also impact the implementation and deployment phases.

Outline of the paper. Section 2 introduces the basics of *AnBx*. Section 3 focuses on the semantics of the language and presents our formal results. Sections 4-6 discuss our case studies. Section 7 concludes the presentation. The *AnBx* implementation, together with its analytical tool and the scripts employed in the case studies, is available online[2].

New contents. This paper integrates and extends the results reported in [33] (first definition of *AnBx*), [4] (formal semantics of the *AnB* language) and [22] (ideal behaviour and cryptographic implementation of secure channels). Section 3 is novel: in previous work the semantics of *AnBx* was defined by a direct translation to *AnB*, based on a cryptographic implementation. Here we recast our cryptographic implementation within the AVISPA Intermediate Format *IF*, and provide an alternative *IF* characterization, based on the ideal channel behaviour. We then prove that the cryptographic implementation conforms with the ideal semantics. Besides representing a valuable theoretical contribution, the semantics correspondence has practical value, as it makes both characterizations equally viable for automatic analysis within any verification framework supporting *IF*. The *SET* case study in Section 6 is new.

[2] `http://www.dais.unive.it/~modesti/anbx/`

Protocol : *Diffie-Hellman*
Types :
  Agent A, B;
  Number g, X, Y, Msg;
Knowledge :
  A : A, B, g;
  B : A, B, g;
Actions :
  $A \to B, (A \mid B \mid -)$: $\mathsf{exp}(g, X)$
  $B \to A, (B \mid A \mid -)$: $\mathsf{exp}(g, Y)$
  $A \to B, (- \mid - \mid -)$: $\{\mid A, Msg \mid\}_{\mathsf{exp}(\mathsf{exp}(g,Y),X)}$
Goals :
  B authenticates A on Msg
  Msg secret between A, B

Figure 1: Diffie-Hellman specification in *AnBx*

2. *AnBx* Protocol Specifications

AnBx is a formal protocol specification language based on the popular (informal) *Alice & Bob* notation. *AnBx* conservatively extends the *AnB* specification language [4] with a richer notion of communication channel.

2.1. Protocol Types and Agent Knowledge

Protocol narrations in *AnBx* are built around an underlying signature of typed identifiers that include protocol variables, constants, and function symbols. Variables are noted with uppercase initials and represent values that are determined dynamically, at each protocol run. Constants, in turn, are noted by lower-case identifiers and represent values and functions that are invariant across different protocol executions. As an example, consider the *AnBx* specification of the Diffie-Hellman key exchange protocol in Figure 1. Variables of type Agent are *roles*: here we have the roles A and B, which get instantiated to arbitrary concrete agents when executing the protocol. The numbers g, X and Y, in turn, are the (constant) group generator and the (variable) random exponents of the Diffie-Hellman key exchange.

For each role, the protocol specification describes the knowledge that an agent playing

that role needs to execute the protocol: this indirectly specifies what the intruder will know when playing one of the roles of the protocol. Only variables of type Agent may be part of the initial knowledge. All other variables represent values that are chosen randomly by the participant who first uses them, e.g., in the example A chooses X and B chooses Y.

2.2. Protocol Actions

The core of an *AnBx* specification consists of the message exchanges between the participants in an ideal, unattacked run of the protocol. Every action has either of the two forms below:

$$A \to B, \eta : M \qquad \text{or} \qquad A \overset{@}{\to} B, \eta : M,$$

noting standard and *fresh* exchanges, respectively. In both cases, an agent playing role A communicates message M to the agent playing role B, along a communication channel that conveys the security guarantees specified by the *exchange mode* η. The *AnBx* modes are triples:

$$(Auth|\, Verifiers|\, Conf),$$

whose components may be set to an agent name (a list of names for the *Verifiers* field), or unset, in which case they are filled with the distinguished symbol "$-$". When the *Conf* field is set, the action represents a confidential exchange, which guarantees that only the agent named in the field has access to the message. When the *Auth* field is set, the action identifies an authentic exchange, which guarantees that the message originates from the agent named in the field; the *Verifiers* field must be set if and only if the *Auth* field is set, to include a nonempty list of agents that are entitled to verify the authenticity of the message. Authentic exchanges may further specify that the message being exchanged is *freshly* communicated by the agent referenced in the *Auth* field: the notation $A \overset{@}{\to} B, \eta : M$ serves that purpose. None of the modes conveys any guarantee that the intended recipients will eventually receive the message.

Though the intended purpose of the channel modes is to hide low-level communication details, we remark that *AnBx* conservatively extends the *AnB* notation, making it possible to freely intermix abstract exchanges and cryptographic terms. Note, in particular, that the first two actions in the Diffie-Hellman specification in Figure 1 employ the channel modes to express the authentic exchange of the two "half keys", while the third describes the exchange of message *Msg* encrypted under the new key.

The idea to structure protocol specifications around abstract mechanisms for secure communications is certainly not new, as we discussed in Section 1. Among the various approaches in the literature, the closest to ours is the "bullet" notation supported by *AnB•* [22], a specification language providing support for confidential and authenticated channels. Every exchange mode available in *AnB•* can be easily encoded in *AnBx*, as shown in Table 1 below; however, *AnBx* provides additional expressiveness, as we discuss in the next section.

	AnB•	*AnBx*
PLAIN	$A \to B$	$A \to B, (-\,|\,-\,|\,-)$
AUTHENTIC	$A \bullet\!\!\to B$	$A \to B, (A\,|\,B\,|\,-)$
CONFIDENTIAL	$A \to\!\!\bullet B$	$A \to B, (-\,|\,-\,|\,B)$
SECURE	$A \bullet\!\!\to\!\!\bullet B$	$A \to B, (A\,|\,B\,|\,B)$

Table 1: Encoding of *AnB•* in *AnBx*

2.3. Forwarding Modes

In addition to the standard *AnB•* exchanges, the *AnBx* modes allow additional generality. Specifically, *AnBx* provides primitive support for message *forwarding*, a feature which is not offered by existing proposals, but constitutes a recurrent communication pattern in practical applications. We will provide examples of concrete uses of forwarding in our case studies; for the moment, we just illustrate the concept with some simple examples.

The first example shows how authenticity can be preserved upon forwarding:

$$\begin{aligned} A \to B, (A\,|\,B,C\,|\,-): &\quad M \\ B \to C, (A\,|\,B,C\,|\,-): &\quad M \end{aligned}$$

The first action denotes an authentic exchange that originates from A and is meant to be delivered to both B and C. Upon receiving M, agent B forwards it to C in the second action, preserving the authenticity guarantees by A. Notice that the mode $(A\,|\,B,C\,|\,-)$ in the second exchange still mentions A as the source of the communication, even though the message is sent by B. This pattern cannot be encoded in the *AnB•* notation, since authentic messages are always

assumed to be originated by the agent specified on the tail of the arrow.

Forwarding modes can be used also to implement a form of "blind" delivery, arising when an agent relays messages that are intended to remain confidential for a third party:

$$A \to B, (-|-|C): \quad M$$
$$B \to C, (-|-|C): \quad M$$

Here, A sends M to C confidentially, relying on B to deliver the message. As in the previous case, this protocol cannot be expressed in the $AnB\bullet$ notation, in this case because secret messages are always intended to be disclosed to the agent specified on the head of the arrow.

Message forwarding is also available for fresh exchanges, in various combinations. Assume message M is sent freshly from A to B:

$$A \overset{@}{\to} B, (A|B,C|-): \quad M$$

Then both the following actions:

$$B \ \to \ C, (A|B,C|-) : M$$

and:

$$B \ \overset{@}{\to} \ C, (A|B,C|-) : M$$

are legal. With the first action, M is forwarded to C without any freshness guarantee, whereas the second action allows C to verify the freshness of the transmission.

2.4. Protocol Goals

AnBx protocol specifications are analyzed and validated against a set of security *goals*, that specify the properties expected of the protocol. Like its predecessors, *AnBx* supports three standard kinds of goals, which we briefly review below, referring the reader to [4] for full details.

- *Weak Authentication* goals have the form:

  $$B \text{ weakly authenticates } A \text{ on } M,$$

  and are defined in terms of non-injective agreement on the runs of the protocol [34];

- *Authentication* goals have the form:

  $$B \text{ authenticates } A \text{ on } M,$$

  and are defined in terms of injective agreement on the runs of the protocol, guaranteeing the freshness of the exchange;

- *Secrecy* goals have the form:

  $$M \text{ secret between } A_1, \ldots, A_k,$$

  and are intended to specify which agents are entitled to learn message M at the end of the protocol run.

3. *AnBx* Semantics

Following previous proposals [4, 22], we define the semantics of *AnBx* in terms of a translation to the AVISPA Intermediate Format *IF* [35]. *IF* is a set-rewriting calculus in which the semantics of a protocol is described in terms of a set of facts that encode the knowledge of the honest agents and the intruder at the different protocol steps, and a set of rewriting rules describing the state transitions of the participants and the intruder during the protocol execution. The rewriting rules for honest participants are generated from the *AnBx* protocol specification, while the capabilities available to the intruder are modelled by protocol-independent rules, i.e., the intruder is not forced to follow the protocol specification.

We define the translation from *AnBx* to *IF* in several steps (Figure 2), conveniently exploiting the existing AnB2IF compiler [4] as a black box. Given an *AnBx* specification, we translate it into a corresponding *AnB* specification, in which the *AnBx* modes are expressed as message *tags* (Section 3.1). The resulting *AnB* specification is fed to the AnB2IF compiler, which extracts from the narration the actions associated with the protocol agents, and renders them as *IF* rewriting rules (Section 3.2). The resulting *IF* rules still include the tags from the annotated *AnB* narration: a further transformation step (Sections 3.3 and 3.4) completes the translation, exploiting the tags to produce a *cryptographic IF* specification and an *ideal IF* specification. We refer to these two constructions as the Cryptographic Channel Model (*CCM*) and the Ideal Channel Model (*ICM*) respectively. The two models are contrasted and related in Section 3.5.

3.1. From AnBx to AnB

The first step of the translation transforms each action in the *AnBx* narration into a corresponding *AnB* action bearing additional annotations, which drive the later stages of the translation.

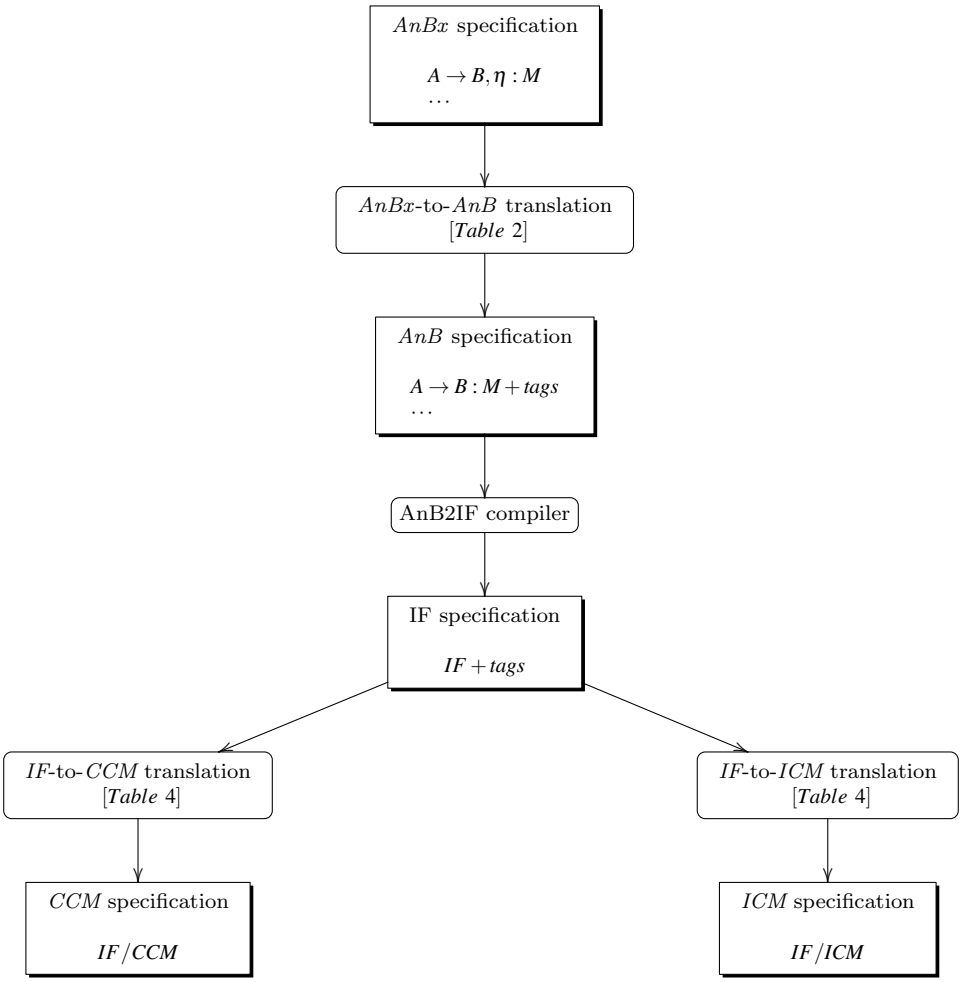

Figure 2: Translation from *AnBx* to *IF*/*CCM* and *IF*/*ICM*.

The *AnBx*-to-*AnB* translation is conceptually simple, though the presence of the fresh modes and their interaction with the forward modes hide a few subtleties. Our characterization of freshness relies on a simple mechanism by which the sender generates a fresh nonce and the recipient caches every nonce it receives, telling fresh messages from replicas by checking whether the received nonce is in the cache. In case of forwarding of a fresh message, we reuse the same nonce generated at the step which introduced the message being forwarded.

In order to ensure that newly generated nonces are indeed fresh, the *AnBx*-to-*AnB* translation keeps track in a store ξ of all the protocol variables introduced to represent the different nonces created along the protocol steps. Formally, the store is a partial function from triples of the form $(A, \tilde{V}, M)$ to nonce variables N. The store ξ is used in the translation to *AnB* (Table 2), as described below:

- At each fresh exchange which is not a forward, we first select a nonce variable N that does not occur in the range of ξ and then we let $\xi(A, \tilde{V}, M) = N$, where A is the name of the source agent, $\tilde{V}$ is the (non-empty) list of verifiers and M is the message exchanged in the *AnBx* specification; this information enables the reuse of N in all the possible future forwards of message M;

- At each authentic or secure forward action, we lookup the domain of ξ in search of a triple matching the *Auth* and the *Verifiers* components of the mode, as well as the message being forwarded; if such a triple exists, the action is a forward of a fresh exchange, and we include the corresponding

$$
\begin{array}{lll}
[\![A \to B, (-|-|-) : M]\!]_\xi = & A \to B : \mathsf{plain}, M & \\
[\![A \to B, (-|-|\hat{B}) : M]\!]_\xi = & A \to B : \mathsf{ctag}, \mathsf{blind}_{\hat{B}}(M) & \\
[\![A \to B, (\hat{A}\,|\,\tilde{V}\,|-) : M]\!]_\xi = & A \to B : \mathsf{atag}, \hat{A}, \tilde{V}, M, N & \text{if } \hat{A} \neq A \text{ and } \xi(\hat{A}, \tilde{V}, M) = N \\
= & A \to B : \mathsf{atag}, \hat{A}, \tilde{V}, M & \text{otherwise} \\
[\![A \to B, (\hat{A}\,|\,\tilde{V}\,|\,\hat{B}) : M]\!]_\xi = & A \to B : \mathsf{stag}, \mathsf{blind}_{\hat{B}}(\hat{A}, \tilde{V}, M, N) & \text{if } \hat{A} \neq A \text{ and } \xi(\hat{A}, \tilde{V}, M) = N \\
= & A \to B : \mathsf{stag}, \mathsf{blind}_{\hat{B}}(\hat{A}, \tilde{V}, M) & \text{otherwise} \\
[\![A \xrightarrow{@} B, (\hat{A}\,|\,\tilde{V}\,|-) : M]\!]_\xi = & A \to B : \mathsf{fatag}, \hat{A}, \tilde{V}, M, N & \text{if } \hat{A} = A \text{ (with } N \text{ chosen fresh in } \xi) \\
 & & \text{or } \hat{A} \neq A \text{ and } \xi(\hat{A}, \tilde{V}, M) = N \\
[\![A \xrightarrow{@} B, (\hat{A}\,|\,\tilde{V}\,|\,\hat{B}) : M]\!]_\xi = & A \to B : \mathsf{fstag}, \mathsf{blind}_{\hat{B}}(\hat{A}, \tilde{V}, M, N) & \text{if } \hat{A} = A \text{ (with } N \text{ chosen fresh in } \xi) \\
 & & \text{or } \hat{A} \neq A \text{ and } \xi(\hat{A}, \tilde{V}, M) = N
\end{array}
$$

Table 2: Translation from *AnBx* to *AnB*

nonce from ξ among the components of the forwarded message, irrespective of whether the forward is fresh or not (this choice is technically convenient in the definition of the translation). If the triple does not belong to the domain of ξ, then the source action must be non-fresh, thus no nonce is included in the forward of the generated message.

- The translation is undefined when a fresh forward is performed, but no matching triple is found in the domain of ξ.

In a practical implementation, one would of course either use timestamps or sequence numbers, in order to limit the amount of data that the receiver has to store. We remark, however, that these realizations are essentially equivalent to our formal model[3].

A further subtlety in the translation arises from blind forwards, i.e., when the recipient A of a message M differs from the final intended receiver B, and the message M should not be exposed to A. To capture the desired effect, we wrap M inside the constructor blind_B, to denote that it should be readable only by B.

The translation clauses are listed in Table 2, where we do not explicitly track the updating of ξ for the sake of readability. The tags plain, ctag, atag and stag are just as in [22]: in addition, we include the new tags fatag and fstag to account for freshly authenticated channels. All the tags are public constants in the target *AnB* specification and blind_X is a function symbol available to every agent (including the intruder) for any X. The specification is also extended with private function symbols $\mathsf{unblind}_X$, parameterized over the agents identity, which are used to extract the confidential messages[4].

As a first, simple illustration, below we give the annotated *AnB* narration that results from applying the translation to the *AnBx* specification of the protocol in Figure 1:

$$
\begin{array}{ll}
A \to B: & \mathsf{atag}, A, B, \mathsf{exp}(g, X) \\
B \to A: & \mathsf{atag}, B, A, \mathsf{exp}(g, Y) \\
A \to B: & \mathsf{plain}, \{|Msg|\}_{\mathsf{exp}(\mathsf{exp}(g,Y),X)}
\end{array}
$$

As a further example, consider the following variant of the blind forward protocol examined earlier on:

$$
\begin{array}{ll}
A \to B, (-|-|C): & Msg, token \\
B \to C, (-|-|C): & Msg, token
\end{array}
$$

where we assume that the three agents use *token* as a known tag marking their exchanges. The resulting *AnB* narration is as follows:

$$
\begin{array}{ll}
A \to B: & \mathsf{ctag}, \mathsf{blind}_C(Msg, token) \\
B \to C: & \mathsf{ctag}, \mathsf{blind}_C(Msg, token)
\end{array}
$$

[3] A further possible alternative is to use challenge-response protocols, but these generate additional network traffic, which in turn would considerably complicate our exposition of the two channel models and their relationship, as well as the practical model-checking problems induced in our tool.

[4] In our implementation we actually rely on the OFMC facility for asymmetric cryptography, since the current implementation of AnB2IF does not support user-defined algebraic theories. Namely, we let $\mathsf{blind}_X(M) \triangleq \{M\}_{\mathsf{b}(X)}$ where $\mathsf{b}(\cdot)$ is a public function symbol and $\mathsf{inv}(\mathsf{b}(X))$ is known only to X.

Error conditions. If none of the clauses in Table 2 applies, the translation is undefined and an error is reported. Errors signal *unexecutable* specifications, which expect the protocol participants to send messages they are unable to compose, since they lack some of the required information bits. One such error condition arises when an agent is expected to execute a fresh forward action for a message it received without any freshness guarantee, as in the following specification:

$$A \to B, (A \,|\, B, C \,|\, -) : \quad M$$
$$B \xrightarrow{@} C, (A \,|\, B, C \,|\, -) : \quad M$$

Further cases of unexecutable specifications are identified by a subsequent translation step, specifically during the *AnB*-to-*IF* translation. Indeed, the $\mathsf{blind}_X(M)$ construction for confidential messages has precisely the purpose to signal to the AnB2IF compiler that message M can only be seen by X, so that a protocol turns out to be unexecutable if such a blinded message needs to be read by another agent. Consequently, a sequence of *AnBx* actions like the one below is translated successfully to *AnB*, but the AnB2IF compiler will reject it as non-executable, since after the first exchange B has access to $\mathsf{blind}_C(M)$ but not to M:

$$A \to B, (- \,|\, - \,|\, C) : \quad M$$
$$B \to C, (- \,|\, - \,|\, -) : \quad M$$

3.2. From AnB to IF

The AVISPA Intermediate Format *IF* [35] is a low-level language for specifying transition systems using set rewriting. We refer the reader to [4] for full details on the translation from *AnB* to *IF*; here, we just provide an informal overview to make the paper self-contained.

An *IF* specification $P = (I, R, G)$ consists of an initial state I, a set of transition rules R for the protocol participants and the intruder, and a set of goals G that determine which states count as attack states. Our notion of attack state coincides with the one used in standard *AnB* [36], defining violations to secrecy and authentication (in terms of injective or non-injective agreement). A protocol is *safe* when no attack state is reachable from the initial state using the transition rules.

An *IF* state is a set of ground facts, separated by dots ("."), which encode the knowledge of the different protocol agents. We distinguish two kinds of facts: $\mathsf{ik}(m)$, which denotes that the intruder knows the term m, and $\mathsf{state}_{\mathscr{A}}(A, m_1, \ldots, m_n)$, which characterizes the local state of an honest agent during the protocol execution by the terms A, $m_1, \ldots, m_n$. The constant $\mathscr{A}$ identifies the role of the agent, and, by convention, the first message A denotes the name of that agent[5]. Our formalization of the intruder also includes a further class of facts of the form $\mathsf{dishonest}(A)$ to identify the dishonest agents participating in the protocols. While many tools assume that there is only a single dishonest agent i (the "intruder"), our model supports any number of collaborating dishonest agents – one may still think of one intruder who has compromised several agents and can now use their identities.

We now discuss how the *initial state* is generated from an *AnB* specification. Let n denote a bounded number of protocol sessions and let $\sigma_1, \ldots, \sigma_n$ be corresponding mappings from the protocol roles $R_1, \ldots, R_m$ to concrete agent names. Let K_j stand for the initial knowledge of the role R_j, then the initial state is:

$$\bigcup_{1 \le i \le n, 1 \le j \le m} \begin{cases} \{\mathsf{state}_{\mathscr{R}_j}(K_j\sigma_i)\} & \text{if } R_j\sigma_i \neq \mathsf{i} \\ \{\mathsf{ik}(K_j\sigma_i), \mathsf{dishonest}(\mathsf{i})\} & \text{if } R_j\sigma_i = \mathsf{i} \end{cases}$$

where i is a reserved constant denoting the identity of the intruder. The initial state thus consists of the local states of the honest agents and the initial knowledge of the intruder, which is determined by the compromised agents; a $\mathsf{dishonest}(\mathsf{i})$ fact is introduced when at least one of the agents is compromised.

The *transition rules* of an *IF* specification are of the form $L \mid \mathit{Cond} = [\mathscr{X}] \Rightarrow R$ where L and R are states, $\mathscr{X}$ is a set of fresh variables (representing fresh values generated at run-time), and Cond is a set of conditions, expressed as (in)equalities and negated predicates. The semantics of an *IF* rule is defined by the state transitions it enables: from a state S the rule enables a transition to a state S' iff there exists a substitution σ of the variables of L and $\mathscr{X}$ such that $L\sigma \subseteq S$, $S' = (S \setminus L\sigma) \cup R\sigma$, and $\mathscr{X}\sigma$ are fresh constants not occurring in S; moreover, the conditions $\mathit{Cond}\sigma\tau$ are true in S for every substitution τ of the variables in Cond that do not occur in L. We assume the $\mathsf{ik}(\cdot)$ and the $\mathsf{dishonest}(\cdot)$ facts to be *persistent*, i.e., to be al-

[5] In contrast to the convention used in the *AnBx* specification, *IF* makes a clear distinction between role names, noted by calligraphic letters such as $\mathscr{A}$, and variables of type Agent.

ways propagated to the right-hand side of any transition.

The semantics of *AnB* is just defined by the translation from an *AnB* specification to *IF*. The main point of the translation is to define the behavior of the *honest* agents in terms of *IF* transition rules, by identifying in particular what checks must be performed on the messages they receive, and how they construct the messages they send out. The behavior of the *intruder*, in contrast, is defined by protocol-independent rules modelling a Dolev-Yao attacker, as in Table 3. We assume the existence of a set of function symbols with an associated arity, which is partitioned into two sets of *public* and *private* symbols respectively.

$$\begin{aligned}
\mathsf{ik}(M).\mathsf{ik}(K) &\Rightarrow \mathsf{ik}(\{M\}_K)\\
\mathsf{ik}(\{M\}_K).\mathsf{ik}(\mathsf{inv}(K)) &\Rightarrow \mathsf{ik}(M)\\
\mathsf{ik}(\{M\}_{\mathsf{inv}(K)}) &\Rightarrow \mathsf{ik}(M)\\
\mathsf{ik}(M).\mathsf{ik}(N) &\Rightarrow \mathsf{ik}(M,N)\\
\mathsf{ik}(M,N) &\Rightarrow \mathsf{ik}(M).\mathsf{ik}(N)\\
\mathsf{ik}(M_1).\cdots.\mathsf{ik}(M_n) &\Rightarrow \mathsf{ik}(f(M_1,\ldots,M_n))
\end{aligned}$$

Table 3: Dolev-Yao intruder rules

The first rule describes both asymmetric encryption and signing, while the second one expresses that the payload of a ciphertext can be retrieved if the corresponding decryption key is known. We use $\mathsf{inv}(\cdot)$ as a *private* function symbol, employed, e.g., to represent the secret component of a given key-pair. The third rule allows the attacker to learn the payload of any signed message he knows. Then, we have rules for tupling and projecting tuple elements, as well as a rule for applying *public* function symbols to known messages (while respecting their arity). We treat constants, including agent identities, as public functions with 0-arity. In this phase of the translation, all the messages exchanged by honest agents are always assumed to be mediated by the intruder, i.e., every communication happens through $\mathsf{ik}(\cdot)$ facts.

We illustrate the translation from *AnB* to *IF* with an example. Specifically, we give the *IF* transition rules for roles *A* and *B* from the *AnB* translation of the protocol in Figure 1. The *IF* transition rules are in Figure 3 below, where for the sake of readability we do not explicitly represent the public tags in the state facts and we turn the side-conditions of the rules into pattern matching:

$$\begin{aligned}
&\mathsf{state}_{\mathscr{A}}(A,B,g) = [X] \Rightarrow\\
&\mathsf{state}_{\mathscr{A}}(A,B,g,X).\mathsf{ik}(\mathsf{atag},A,B,\mathsf{exp}(g,X))\\
&\\
&\mathsf{state}_{\mathscr{A}}(A,B,g,X).\mathsf{ik}(\mathsf{atag},B,A,GY) = [Msg] \Rightarrow\\
&\mathsf{state}_{\mathscr{A}}(A,B,g,X,GY,Msg).\\
&\mathsf{ik}(\mathsf{plain},\{|A,Msg|\}_{\mathsf{exp}(GY,X)})\\
&\\
&\mathsf{state}_{\mathscr{B}}(B,A,g).\mathsf{ik}(\mathsf{atag},A,B,GX) = [Y] \Rightarrow\\
&\mathsf{state}_{\mathscr{B}}(B,A,g,GX,Y).\mathsf{ik}(\mathsf{atag},B,A,\mathsf{exp}(g,Y))\\
&\\
&\mathsf{state}_{\mathscr{B}}(B,A,g,GX,Y).\\
&\mathsf{ik}(\mathsf{plain},\{|A,Msg|\}_{\mathsf{exp}(GX,Y)}) \Rightarrow\\
&\mathsf{state}_{\mathscr{B}}(B,A,g,GX,Y,Msg)
\end{aligned}$$

Figure 3: *IF* translation of the example of Figure 1

Notice in the second clause that *A* accepts any value *GY* from the network, not necessarily the result of a correct Diffie-Hellman exponentiation, and applies it to encrypt the last message of the protocol. Conversely, in the fourth clause, *B* checks that the first encrypted message component is indeed the identity of *A*, but it cannot check anything about *Msg*, since it is freshly generated by another participant.

3.3. From IF to CCM

The Cryptographic Channel Model realizes the *AnBx* channel modes by means of digital signatures and public-key encryptions, represented in a simple symbolic model of cryptography.

Honest agents. The translation of the honest agents is based on the *IF*-to-*CCM* mapping defined in Table 4. For rules generated by the AnB2IF compiler, the corresponding *CCM* rule results from applying the mapping *IF*-*CCM* in the table to the intruder facts. In the *CCM* code, we additionally associate two key-pairs $(\mathsf{pk}(A),\mathsf{inv}(\mathsf{pk}(A)))$ for encryption/decryption, and $(\mathsf{sk}(A),\mathsf{inv}(\mathsf{sk}(A)))$ for verification/signing with every agent *A* acting as the target of a confidential exchange or as the source of an authentic message. These keys are only used for encoding channels and must not appear in the *AnBx* protocol specification.

The message *M* occurring in all clauses in Table 4 may be an arbitrary message. The last clause is the exception, as it only applies to variables: this clause handles the case of agents that are expected to execute blind forward actions

IF	*CCM*	*ICM*
$\mathsf{ik}(\mathsf{plain},M)$	$\mathsf{ik}(M)$	$\mathsf{ik}(M)$
$\mathsf{ik}(\mathsf{ctag},\mathsf{blind}_B(M))$	$\mathsf{ik}(\{M\}_{\mathsf{pk}(B)})$	$\mathsf{cnfCh}(B;M)$
$\mathsf{ik}(\mathsf{atag},A,\tilde{V},M)$	$\mathsf{ik}(\{\tilde{V},M\}_{\mathsf{inv}(\mathsf{sk}(A))})$	$\mathsf{athCh}(A;\tilde{V};M)$
$\mathsf{ik}(\mathsf{stag},\mathsf{blind}_B(A,\tilde{V},M))$	$\mathsf{ik}(\{\{\tilde{V},M\}_{\mathsf{inv}(\mathsf{sk}(A))}\}_{\mathsf{pk}(B)})$	$\mathsf{secCh}(A;\tilde{V};B;M)$
$\mathsf{ik}(\mathsf{fatag},A,\tilde{V},M,N)$	$\mathsf{ik}(\{\tilde{V},M,N\}_{\mathsf{inv}(\mathsf{sk}(A))})$	$\mathsf{athCh}(A;\tilde{V};M,N)$
$\mathsf{ik}(\mathsf{fstag},\mathsf{blind}_B(A,\tilde{V},M,N))$	$\mathsf{ik}(\{\{\tilde{V},M,N\}_{\mathsf{inv}(\mathsf{sk}(A))}\}_{\mathsf{pk}(B)})$	$\mathsf{secCh}(A;\tilde{V};B;M,N)$
$\mathsf{ik}(t,X)$ $t\in\{\mathsf{ctag},\mathsf{stag},\mathsf{fstag}\}$, X variable	$\mathsf{ik}(X)$	$\mathsf{ik}(X)$

Table 4: Translation from *IF* to *CCM* and *ICM*

for confidential or (fresh) secure messages. In the AnB2IF translation, such agents receive the messages to be forwarded as terms of the form (t,X) for some variable X, as they are going to accept any message at such steps, without inspecting it: therefore, to obtain the corresponding *CCM* code, we just remove the tag.

To illustrate this, consider again the annotated *AnB* blind-forward example we examined in Section 3.1:

$$\begin{array}{lcll} A & \to & B: & \mathsf{ctag},\mathsf{blind}_C(Msg,token) \\ B & \to & C: & \mathsf{ctag},\mathsf{blind}_C(Msg,token) \end{array}$$

Though *token* is assumed to be known to all agents, the forward action by B is performed irrespectively of the actual content of the message it receives, since B is not able to perform any check on a confidential message for C. This is shown by the *IF* code produced by the translation of the exchange to the *CCM*:

$$\mathsf{state}_{\mathscr{A}}(A,B,C,token) =[Msg]\Rightarrow \mathsf{state}_{\mathscr{A}}(A,B,C,token,Msg).\mathsf{ik}(\{Msg,token\}_{\mathsf{pk}(C)})$$

$$\mathsf{state}_{\mathscr{B}}(B,C,A,token).\mathsf{ik}(X) \Rightarrow \mathsf{state}_{\mathscr{B}}(B,C,A,token,X)$$

$$\mathsf{state}_{\mathscr{C}}(C,A,B,token).\mathsf{ik}(\{Msg,token\}_{\mathsf{pk}(C)}) \Rightarrow \mathsf{state}_{\mathscr{C}}(C,A,B,token,Msg)$$

In the second transition rule, B accepts every message X provided by the intruder. (Recall that the $\mathsf{ik}(X)$ fact is not repeated explicitly on the right-hand side of the arrow, since such facts are persistent.) Conversely, in the third rule C can verify that the second component of the encryption is indeed the expected *token* available in her knowledge.

An additional measure is needed for translating to the *CCM* the transition rules expecting a fresh message on input. These rules are easily identified in the annotated *AnB* code, as they have an occurrence of fatag/fstag in their incoming message. For any such transition rule, let B be the receiver, and N the nonce associated with the fresh message. Now, to implement the nonce-checking mechanism of replay protection we discussed in Section 3.1, it is enough (*i*) to include the side condition $\mathsf{not}(\mathsf{seen}(B,N))$ in the transition rule, and (*ii*) to introduce the fact $\mathsf{seen}(B,N)$ to the right-hand side of the same rule. For instance, for the sender of the message $A \overset{@}{\to} B,(A\,|\,B\,|-):Msg$, the *CCM* will comprise a transition rule of the form:

$$\ldots =[N]\Rightarrow \mathsf{ik}(\{B,Msg,N\}_{\mathsf{inv}(\mathsf{pk}(A))}).\ldots$$

with N fresh. Correspondingly, on the receiver side, the transition rule in the *CCM* will be structured as follows:

$$\ldots.\mathsf{ik}(\{B,Msg,N\}_{\mathsf{inv}(\mathsf{pk}(A))}) \mid \mathsf{not}(\mathsf{seen}(B,N)) \Rightarrow \mathsf{seen}(B,N).\ldots$$

As a result, message M is received only if the nonce N was never seen before by the receiver: if that is the case, and the message is accepted, the receiver adds N to its cache of seen nonces.

Intruder rules. The intruder rules of the CCM are just the Dolev-Yao intruder rules in Table 3, where we assume that the symbols $\mathsf{sk}(\cdot)$ and $\mathsf{pk}(\cdot)$ introduced earlier on are public functions. Consequently, every agent, including the intruder, can obtain the public keys of every other agent as soon as its name is known (this implies that the intruder knows all the public keys). Since the function $\mathsf{inv}(\cdot)$ providing the ability to construct signing and decryption keys is private, each agent A knows only her own private keys $\mathsf{inv}(\mathsf{sk}(A))$ and $\mathsf{inv}(\mathsf{pk}(A))$. Notice that

private keys of dishonest agents are available to the intruder, according to the definition of the *IF* initial state in Section 3.2.

3.4. From IF to ICM

The Ideal Channel Model provides for a direct representation of the communication modes in terms of corresponding *IF* state facts that encode the types of channel involved in the exchanges. In particular, the ideal semantics draws on the constructors athCh, cnfCh and secCh, around which we define persistent state facts that track the protocol exchanges. Protocol-independent rewriting rules, in turn, characterize the intended behaviour of the ideal channels.

Honest agents. The translation of the honest agents is based on the *IF*-to-*ICM* mapping defined in Table 4. For each rule generated by the AnB2IF compiler, the corresponding *ICM* rule results from applying the mapping *IF*-*ICM* in the table. Similarly to the *CCM* translation, the last case in the table handles a blindly forwarding agent who cannot check anything about the message being forwarded.

For our blind forwarding example, the translation to the *ICM* generates the following *IF* transition rules:

$$\mathsf{state}_{\mathscr{A}}(A,B,C,token) =[Msg]\Rightarrow \mathsf{state}_{\mathscr{A}}(A,B,C,token,Msg).\mathsf{cnfCh}(C;Msg,token)$$

$$\mathsf{state}_{\mathscr{B}}(B,C,A,token).\mathsf{ik}(X) \Rightarrow \mathsf{state}_{\mathscr{B}}(B,C,A,token,X)$$

$$\mathsf{state}_{\mathscr{C}}(C,A,B,token).\mathsf{cnfCh}(C;Msg,token) \Rightarrow \mathsf{state}_{\mathscr{C}}(C,A,B,token,Msg)$$

The only significant difference with respect to the *CCM* is that the encrypted message for C is replaced by a $\mathsf{cnfCh}(C;\cdot)$ channel fact.

Two comments are in order for the *ICM* translation. First, nonces are implicitly included in the payload of the message when freshness is lost upon forwarding: this choice reflects the corresponding behavior in the *CCM*, where nonces cannot be removed from digitally signed packets. Second, given that the state channel facts employed in the *ICM* are persistent, we need additional measures to protect against replicas in all transition rules expecting a fresh message on input. For that purpose, we rely on the very same mechanism described earlier for the cryptographic model, based on the $\mathsf{seen}(\cdot,\cdot)$ facts to tell replicas apart. Even though we could define additional non-persistent channel facts to model fresh channels, this choice simplifies the definition of the correspondence between the two channel models and the related proof.

Intruder rules. The intruder rules constitute the key component of the ideal semantics, as it is through these rules that we define the actual interpretation of our channel facts. Specifically, in the *ICM*, the Dolev-Yao intruder rules in Table 3 are extended with the rules reported in Table 5 below.

$$\begin{aligned}
\mathsf{ik}(\tilde{V},M).\mathsf{dishonest}(A) &\Rightarrow \mathsf{athCh}(A;\tilde{V};M)\\
\mathsf{athCh}(A;\tilde{V};M) &\Rightarrow \mathsf{ik}(\tilde{V},M)\\
\mathsf{ik}(B).\mathsf{ik}(M) &\Rightarrow \mathsf{cnfCh}(B;M)\\
\mathsf{cnfCh}(B;M).\mathsf{dishonest}(B) &\Rightarrow \mathsf{ik}(M)\\
\mathsf{athCh}(A;\tilde{V};M).\mathsf{ik}(B) &\Rightarrow \mathsf{secCh}(A;\tilde{V};B;M)\\
\mathsf{secCh}(A;\tilde{V};B;M).\mathsf{dishonest}(B) &\Rightarrow \mathsf{athCh}(A;\tilde{V};M)
\end{aligned}$$

Table 5: Intruder rules for *ICM*

An intruder can forge a message over an authentic channel only if the associated sender identity is compromised, while he can learn every message sent over an authentic channel. Dually, an intruder can send over a confidential channel every message he can compose, but he can learn a message sent over a confidential channel only if the associated receiver identity is compromised.

In addition, we give the intruder two more abilities for secure channels, corresponding to those available in the *CCM*. Specifically, the last two transition rules in Table 5 provide the intruder with the ability to secure an authentic channel, and to drop confidentiality from secure channels shared with compromised agents.

The two transition rules reflect the corresponding intruder capabilities available in the *CCM*, where an intruder can upgrade a message on a (fresh) authentic channel to one on a (fresh) secure channel, by encrypting it, and dually access the contents of a secure message directed to a compromised receiver, by decrypting it.

3.5. Relating ICM and CCM

We complete our formalization of the *AnBx* semantics by analyzing the relationship between the *ICM* and the *CCM* characterizations. In

particular, we define and prove correct a semantic equivalence between the two models. As a first step, we define a correspondence relation $\sim$ between *ICM* and *CCM* states, that relates states that only differ in their encoding of channels. Intuitively, two $\sim$-correspondent states encode the same local knowledge for each protocol agent and the intruder. To define this notion easily, we first introduce an *erasure* operation used to remove the cryptographic keys introduced in the CCM encoding.

Definition 1 (Erasure)**.** *Given a CCM state S, let $|S|$ be the CCM state obtained by removing the cryptographic keys introduced in the CCM encoding, i.e., by deleting any element of the form $\mathsf{pk}(A)$, $\mathsf{sk}(A)$, $\mathsf{inv}(\mathsf{pk}(A))$, $\mathsf{inv}(\mathsf{sk}(A))$ from the agents' knowledge (including the intruder).*

The formal definition of $\sim$ is given below: it relies on the simple mapping from *ICM* states to *CCM* states shown in Table 6. Notice that message M may include a nonce in the case of channels providing freshness guarantees.

ICM	*CCM*
$\mathsf{cnfCh}(B;M)$	$\mathsf{ik}(\{M\}_{\mathsf{pk}(B)})$
$\mathsf{athCh}(A;\tilde{V};M)$	$\mathsf{ik}(\{\tilde{V},M\}_{\mathsf{inv}(\mathsf{sk}(A))})$
$\mathsf{secCh}(A;\tilde{V};B;M)$	$\mathsf{ik}(\{\{\tilde{V},M\}_{\mathsf{inv}(\mathsf{sk}(A))}\}_{\mathsf{pk}(B)})$

Table 6: Mapping the *ICM* to the *CCM*

Definition 2 (Corresponding States)**.** *Let S_1 be an **ICM** state and S_2 be a **CCM** state. We say that S_1 and S_2 are* corresponding states *(noted $S_1 \sim S_2$) if and only if there exists a bijection from the facts in S_1 to the facts in $|S_2|$ that is the identity on all but the channel facts and behaves on the channel facts according to the mapping in Table 6.*

Based on this definition, we now turn to the problem of establishing a semantic equivalence between the *ICM* and the *CCM*, proving a one-to-one correspondence between attack states. Given an *AnBx* specification P, let $\mathit{CCM}(P)$ and $\mathit{ICM}(P)$ stand for its translation to the ***CCM*** and to the ***ICM***, respectively.

Theorem 1. *Let P be an AnBx specification. For each state S_1 reachable from $\mathit{ICM}(P)$ there exists a state S_2 reachable from $\mathit{CCM}(P)$ such that $S_1 \sim S_2$.*

Proof. We proceed by induction on the number of steps performed. The initial states are equivalent modulo $\sim$ by definition of our translation. Let us assume, by induction hypothesis, that $S_1 \sim S_2$ for some reachable *ICM* state S_1 and some reachable *CCM* state S_2. Let S_1' be an *ICM* state reachable from S_1 in one step: we show that there exists a *CCM* state S_2' such that S_2' is reachable from S_2 and $S_1' \sim S_2'$.

We proceed by a case analysis on the transition rule r applied to rewrite S_1 into S_1'. The easiest case is when r is an intruder rule, which does not involve any channel fact (e.g., a rule like $\mathsf{ik}(M).\mathsf{ik}(K) \Rightarrow \mathsf{ik}(\{M\}_K)$). In this case the very same rule can be applied also in the *CCM* to obtain an equivalent state. A similar reasoning applies for honest agents rules, given the definition of our translation. The most interesting possibility is when r is an intruder rule involving channel facts.

We show the cases for authentic channels as representative of all other cases:

- Let $r = \mathsf{ik}(\tilde{V},M).\mathsf{dishonest}(A) \Rightarrow \mathsf{athCh}(A;\tilde{V};M)$. Since $S_1 \sim S_2$, the intruder knows $\tilde{V}$ and M also in S_2. The *CCM* encoding of the channel fact on the right side of the rule is $\mathsf{ik}(\{\tilde{V},M\}_{\mathsf{inv}(\mathsf{sk}(A))})$. This term can be constructed by the intruder in the *CCM*, since $\mathsf{dishonest}(A)$ implies that $\mathsf{inv}(\mathsf{sk}(A))$ is known to the intruder. Therefore, there is a reachable state S_2' such that $S_1' \sim S_2'$.

- Let $r = \mathsf{athCh}(A;\tilde{V};M) \Rightarrow \mathsf{ik}(\tilde{V},M)$. Since $S_1 \sim S_2$, the intruder knows in S_2 the *CCM* encoding of the channel fact, i.e., we have $\mathsf{ik}(\{\tilde{V},M\}_{\mathsf{inv}(\mathsf{sk}(A))}) \in S_2$. The intruder can thus learn M and $\tilde{V}$ by verification of the signature, using $\mathsf{sk}(A)$. Therefore, an S_2' with $S_1' \sim S_2'$ is reachable.

The proof for confidential and secure channels proceeds along the same lines. □

Theorem 1 ensures that, if we verify a protocol in the *CCM*, then the protocol is also secure in the *ICM*. The opposite direction, instead, does not hold in general, since there is an unbounded number of reachable *CCM* states which do not have any counterpart in the *ICM*, due to the presence of the cryptographic keys for encoding channels. Still, for verification purposes, we are interested in *attack* states, and we can in fact prove a formal result about them. Carrying out such a proof is challenging, since in principle the intruder can abuse channel encodings inside

CCM states for mounting attacks which would not work in the *ICM*, where such cryptographic messages are not present at all.

The insight is interpreting such abuses as a special case of "type-flaw" attacks, as the intruder is actually fooling the honest agents into improperly using cryptographic material related to the channel encodings. Interestingly, it is well-known that type-flaw attacks can be systematically prevented by good protocol design, when all message components are annotated with sufficient information to enforce a unique interpretation [37, 38]. These "typing results" do not keep the intruder from sending ill-typed messages (e.g., sending an encrypted message in place of a nonce); rather, they ensure that every message (part) has a unique interpretation. Then, it can be shown that if an attack exists, also a well-typed attack exists – hence it never helps the intruder to use ill-typed messages. Considering only well-typed attacks is a convenient proof strategy and it bears no loss of generality for the class of *typeable* protocols.

Typeable protocols. We presuppose a finite set of basic type symbols B (like *nonce*, *agent*, etc.). We define the set T of composed types as the least set that contains B and that is closed under the following property: if $\tau_1, \ldots, \tau_n \in T$ and f is a function symbol of arity n, then also $f(\tau_1, \ldots, \tau_n) \in T$.

We note with Γ typing environments, binding constants and variables of a protocol specification to types, so that $\Gamma(c) \in B$ for every constant c and $\Gamma(X) \in T$ for every variable X. We extend Γ to a function on arbitrary terms as follows:

$$\Gamma(f(t_1, \ldots, t_n)) = f(\Gamma(t_1), \ldots, \Gamma(t_n)).$$

Definition 3 (Typeable Protocol)**.** *Consider a CCM protocol specification P with the standard operators for symmetric and asymmetric encryption, and such that communication occurs only via $\mathsf{ik}(\cdot)$ facts (i.e., the transition rules of the protocol agents operate on disjoint facts for disjoint agents).*

Let the set $MP(P)$ of message patterns *of P be defined as the set of all terms of the form $\mathsf{ik}(m)$ in the initial state and the transition rules of the honest agents; we assume here that variables occurring in $MP(P)$ are α-renamed in such a way that no two distinct elements have a common variable (α-renaming is assumed to be type consistent). Finally, let:*

$$\begin{aligned} SMP(P) \quad = \quad & \{s \mid s \sqsubseteq t \in MP(P) \wedge s \notin \mathcal{V}\} \\ & \cup \{\mathsf{inv}(k) \mid \{m\}_k \sqsubseteq t \in MP(P)\}, \end{aligned}$$

be the the non-variable subterms of message patterns as well as all decryption keys, again under α-renaming ($\sqsubseteq$ denotes the subterm relation and $\mathcal{V}$ is the set of variables).

We say that the protocol P is typeable *in a typing environment Γ if for all $s,t \in SMP(P)$ one has $\Gamma(s) = \Gamma(t)$ whenever s and t have a unifier. We omit Γ when clear from the context.*

Theorem 2. *If there is an attack against a typeable protocol, then there is a well-typed one, i.e., where every variable X is instantiated with a term t such that $\Gamma(X) = \Gamma(t)$.*

Proof. A simple adaptation of the proof in [39]. See Appendix A for details. □

As usual, the notion of typing we adopt rules out as non-typeable many specifications that are actually perfectly alright. This happens when several messages have similar formats. In this case, we cannot apply Theorem 2 regarding well-typed attacks (and invoke the main theorem below). Fortunately, there is a systematic way to make all protocols typeable, by adding tags to tell different messages apart, a practice which is not expensive in the implementation and does not destroy any standard authentication and secrecy property.

We are finally ready to state and prove the result of interest for our typed model. We conjecture that such result may hold true also for arbitrary attacks on any given protocol, but we do not see any viable proof strategy for this more general setting.

Theorem 3. *Let P be an AnBx specification and let us assume a well-typed attack in $CCM(P)$ that leads to the attack state S_2. Then there exists a reachable attack state S_1 in $ICM(P)$ such that $S_1 \sim S_2$.*

Proof. First observe that an honest agent can only receive messages that are a well-typed instance of a message in $MP(P)$ for the *CCM* variant of P. We can thus restrict the intruder to generating only messages (and sub-messages thereof) that honest agents can actually receive or that are the decryption key for a message in his knowledge. These messages are all well-typed instances of $MP(P)$ or $\mathsf{inv}(\cdot)$ thereof.

Further, observe that the key functions $\mathsf{sk}(\cdot)$ and $\mathsf{pk}(\cdot)$ may occur only in the channel encodings in the *CCM* and not in the *AnBx* protocol specification, hence none of the variables in P has a type containing either of these constructors. It is thus enough to assume the intruder only uses the channel keys for composition of messages as it is intended by the protocol, e.g., we can exclude double encryption with the channel key $\mathsf{pk}(\cdot)$, since any other uses of these keys would lead to ill-typed messages.

Now, we prove a stronger statement, namely that any well-typed trace in $\mathit{CCM}(P)$ has a corresponding trace in $\mathit{ICM}(P)$ such that every state in the first trace corresponds (in the sense defined by $\sim$) to some matching state in the second trace. We proceed by induction on the length of the trace. If the trace is empty, then the conclusion is immediate by definition of our translation. Otherwise, assume the trace in $\mathit{CCM}(P)$ includes a transition from a state S_2 to a state S_2'. By inductive hypothesis, there exists a reachable state S_1 in $\mathit{ICM}(P)$ such that $S_1 \sim S_2$. We show that there exists an *ICM* state S_1' such that S_1' is reachable from S_1 and $S_1' \sim S_2'$.

As the most interesting case, consider an asymmetric encryption step of the intruder, encrypting a message M with public key K. We thus have $\{\mathsf{ik}(M), \mathsf{ik}(K)\} \subseteq S_2$, while $\mathsf{ik}(\{M\}_K) \in S_2'$. By the typing assumption, we have either of the following cases:

- Neither M nor K contain $\mathsf{pk}(\cdot)$ or $\mathsf{sk}(\cdot)$ as subterms, i.e., they are not related to channel facts. Then by definition of $\sim$ we have that $\{\mathsf{ik}(M), \mathsf{ik}(K)\} \subseteq S_1$ and so the same step is possible in the *ICM*.

- $K = \mathsf{pk}(B)$ and M does not contain $\mathsf{pk}(\cdot)$ or $\mathsf{sk}(\cdot)$. Then by definition of $\sim$ we have $\mathsf{ik}(M) \in S_1$. The result here corresponds to proving $\mathsf{cnfCh}(B;M) \in S_1'$, which can be generated by the rule $\mathsf{ik}(B).\mathsf{ik}(M) \Rightarrow \mathsf{cnfCh}(B;M)$ in the *ICM*.

- $K = \mathsf{pk}(B)$ and $M = \{\tilde{V}, M_0\}_{\mathsf{inv}(\mathsf{sk}(A))}$, i.e. the intruder turns an authentic message from A for verifiers $\tilde{V}$ into a secure message for B. Since $\mathsf{ik}(M) \in S_2$, by definition of $\sim$ we have $\mathsf{athCh}(A;\tilde{V};M_0) \in S_1$, and thus we can reach the corresponding $\mathsf{secCh}(A;\tilde{V};B;M_0) \in S_1'$ by the rule $\mathsf{athCh}(A;\tilde{V};M).\mathsf{ik}(B) \Rightarrow \mathsf{secCh}(A;\tilde{V};B;M)$ in the *ICM*.

All other encryptions steps would produce messages that cannot be received and we excluded these redundant steps above.

The cases for signing, analysis, and the transitions of honest agents similarly have a correspondence in the *ICM*. □

Theorems 2 and 3 can be combined as follows. Given a protocol P, we verify that its *CCM* translation satisfies the assumptions of the typing result (Theorem 2): note that the conditions to check are purely syntactical and can be mechanized. We then know that, if P has an attack, then it has a well-typed one, so Theorem 3 implies that there is also an attack on the *ICM*. Thus, if $\mathit{ICM}(P)$ is secure, then so is $\mathit{CCM}(P)$.

Conceptually, the *ICM* is the preferential definition of our channels, as it is independent of the specific implementation details and it focuses solely on formalizing the behaviour of channels. This abstract model is more suitable for protocol design. Moreover, for tools like ProVerif [40] and SATMC [41], the ideal model is easier for verification, since it is free of most of the typing problems such as those discussed above. On the other hand, the *CCM* is more convenient in conjunction with other model-checking tools like the ones of AVISPA [27], where *CCM* specifications may be verified directly. Collectively, our results have thus relevant practical consequences for automating security verification with several different tools.

4. Case Study: e-Payment Protocols

We now demonstrate *AnBx* at work on the specification of a wide and interesting class of protocols, namely e-payment protocols.

4.1. Introducing the Case Studies

The first case study we propose is the *iKP* e-payment protocols family, showing how *AnBx* lends itself to a robust and modular design that captures the increasing level of security enforced by the different protocols in the *iKP* family, depending on the number of principals possessing a certified signing key. Interestingly, as a byproduct of our design and verification efforts, we isolate a new flaw in the original *iKP* specification and propose a fix.

The second case study illustrates a revised version of *SET*, a protocol that for its complexity is considered a benchmark for protocol analysis. Here, we shift our attention to some known

security flaws of the protocol and show that our *AnBx* variant is immune to such defects. Notably, the case study employs fresh forward modes to propose a simple solution to a known issue related to payment authorization [42].

In both case studies, our revised versions of the protocols provide stronger security guarantees than the original protocols. This was largely expected, since the *AnBx* channel abstractions convey protection on *all* message components; however, we believe that our exercise of revisiting existing protocols provides evidence about the value of employing adequate channel abstractions for protocol design. In fact, our revised protocols have a much simpler structure than their original specification and, in principle, a robust implementation can be automatically synthesised from their *AnBx* narration, yielding stronger and more scalable security guarantees with limited effort.

We postpone a detailed discussion on the verification setup until Section 4.4, and turn now to the details of the e-payment protocols specification in *AnBx*.

4.2. A Basic e-Payment Scheme

We outline the bare-bone specification of an e-payment protocol, exposing the protocol structure and the message formats common to both our case studies.

We presuppose three principals: a Customer C, a Merchant M and an Acquirer A, i.e., a financial institution entitled to process a payment. In our model, each principal starts with an initial knowledge shared with other participants. Indeed, since most e-payment protocols describe only the payment transaction and do not consider any preliminary phase, we assume that the Customer and the Merchant have already agreed on the details of the transaction, including an order description (*desc*) and a *price*. We also assume that the Acquirer shares with the Customer a customer's account number (*can*) comprising a credit card number and the related PIN. The initial knowledge of the three parties can thus be summarized as follows: C knows *price, desc, can*; M knows *price, desc*; and A knows *can*.

The transaction can be decomposed into the following steps:

1. $C \rightarrow M$: *Initiate*
2. $C \leftarrow M$: *Invoice*
   (In steps 1 and 2 the Customer and the Merchant exchange all the information which is necessary to compose the next payment messages.)
3. $C \rightarrow M$: *Payment Request*
4. $M \rightarrow A$: *Authorization Request*
   (In steps 3 and 4 the Customer sends a payment request to the Merchant. The Merchant uses this information to compose an authorization request for the Acquirer and tries to collect the payment.)
5. $M \leftarrow A$: *Authorization Response*
6. $C \leftarrow M$: *Confirm*
   (In steps 5 and 6 the Acquirer processes the transaction information, and then relays the purchase data directly to the issuing bank, which then authorizes the sale in accordance with the Customer's account. This interaction is not part of the narration. The Acquirer returns a response to the Merchant, indicating success or failure of the transaction. The Merchant then informs the Customer about the outcome.)

Interestingly, steps (4) and (6) involve forwarding operations, since the Customer never communicates directly with the Acquirer, but some credit-card information from the Customer must flow to the Acquirer through the Merchant to compose a reasonable payment request, while the final response from the Acquirer must flow to the Customer through the Merchant to provide evidence of the transaction. Steps (4) and (6) cannot thus be expressed in existing protocol narration frameworks without sacrificing the adoption of their channel abstractions: this prevents a clean, abstract specification of protocols like *iKP* and *SET*.

In addition to some elements of the initial knowledge, other information needs to be exchanged in the previous protocol template. First, to make transactions univocally identifiable, the Merchant generates a fresh transaction ID (*tid*) for each transaction. Second, the Merchant associates to the transaction also a *date* or any appropriate timestamp. Both pieces of information must be communicated to the other parties. The transaction is then defined by a *contract*, which comprises most of the previous information. If Customer and Merchant reach an agreement on it, and they can prove this to the Acquirer, then the transaction can be completed successfully. The details on the structure of the contract vary among different protocols.

At the end of the transaction, the authorization *auth* is then returned by the Acquirer, and communicated to the two other participants.

Message formats. Our protocol templates presuppose the exchange of three kinds of messages: either simple names, m, or tuples of messages $(\tilde{M})$, or else *message digests*.

We represent digest creation simply as a term $[M]$ by which an agent may prove the knowledge of a message M without leaking it to the recipient, e.g., via a hash function: this is modelled through a non-invertible function symbol. We also consider digests which are resistant to dictionary attacks, hence presupposing an implementation based on a hashing scheme that combines the message M with a key known by the principal which must verify the digest. We note with $[M{:}A]$ a digest of a message M which is intended to be verified by A. The symbolic implementation of this HMAC primitive is standard, and full details can be found in the scripts employed for our case studies.

4.3. Protocol Goals

We provide a brief overview of our security properties of interest for e-payment protocols. Further details about the validated protocol goals are later reported for each case study.

A first goal we would like to meet for an e-payment system is that all the principals agree on the contract they sign. In terms of OFMC goals, this corresponds to requiring that each participant can authenticate the other two parties on the *contract*. Moreover, the Acquirer should be able to prove to the other two parties that the payment has indeed been authorized and the associated transaction performed: in OFMC this can be represented by requiring that M and C can authenticate A on the authorization *auth*.

A stronger variant of the goals described above requires that, after completion of a transaction, each participant is able to provide a non-repudiable proof of the effective agreement by the other two parties on the terms of the transaction. In principle, each principal may wish to have sufficient proofs to convince an external verifier that the transaction was actually carried out as she claims. The lack of some of these provable authorizations does not necessarily make the protocol insecure, but it makes disputes between the parties difficult to settle, requiring to rely on evidence provided by other parties or to collect off-line information.

Finally, we are also interested in some secrecy goals, like verifying that the Customer's credit card information *can* is kept confidential, and transmitted only to the Acquirer. In general, we would like to keep the data exchanged by the principals secret among the parties who strictly need to access them for protocol functionality.

4.4. Experimental Setup and Performance

We verified the *AnBx* specifications of *iKP* and *SET* by compiling them into their cryptographic implementation, using our tool, and running OFMC [27] on the generated *CCM* translation against the described security goals. We also encoded and verified the original versions of *iKP* and *SET*, and compared the results with those of the revised versions.

For all the tests we ran OFMC with one and two symbolic sessions. This bounds how many protocol executions the honest agents can engage in, while the intruder is left unbounded thanks to the symbolic lazy intruder technique in OFMC. In the following we say that a goal is met only if it is satisfied in all the considered settings. With two sessions we were unable to complete the full verification due to search space explosion. Therefore, we report (Table 8 and 10) the performance results for the highest depth of the search space we were able to complete for all protocols within the limits of RAM available[6].

Our experiments show that the revised versions of *iKP*, for a given depth of search, can be verified much faster than the original ones, while for *SET* the verification times for the original and revised versions are similar.

Another aspect we consider is the execution speed of a full run of the protocols. We built Java implementations of these protocols automatically generating them with the *AnBx* compiler [11, 12]. On both the original and revised versions we used the same cryptographic primitives and settings[7]. The results of the original and revised versions are usually similar, though the original *3KP* and *signed SET* run slightly faster. However, they are less secure than their revised counterparts.

[6] Configuration - RAM: 8 Gb, CPU: Intel Core i7-4700HQ 2.40 GHz, OS: Windows 8.1

[7] Configuration (JDK8u66) - Symmetric Encryption: AES-128, Asymmetric Encryption: RSA-2048, Hashing: SHA-1, HMAC: HmacSHA1

5. The iKP Protocol Family

The *iKP* protocol family was developed at IBM Research [28, 29, 43] to support credit card-based transactions between customers and merchants (under the assumption that payment clearing and authorization may be handled securely off-line). All protocols in the family are based on public-key cryptography. The idea is that, depending on the number of parties that own certified public key-pairs, we can achieve increasing levels of security, as reflected by the name of the different protocols ($1KP$, $2KP$, and $3KP$).

5.1. Protocol Narration

Despite the complexity of *iKP*, by abstracting from cryptographic details, we can isolate a common communication pattern underlying all the protocols of the family. Namely, a common template can be specified as follows:

1. $C \rightarrow M, \eta_1 : [can{:}A], [desc{:}M]$
2. $C \leftarrow M, \eta_2 : price, tid, date, [contract]$
3. $C \rightarrow M, \eta_3 : price, tid, can, [can{:}A], [contract]$
4. $M \rightarrow A$ (decomposed into two steps to specify different communication modes)
   (a) $M \rightarrow A, \eta_{4a} :$ $price, tid, can, [can{:}A], [contract]$
   (b) $M \rightarrow A, \eta_{4b} :$ $price, tid, date, [desc{:}M], [contract]$
5. $M \leftarrow A, \eta_5 : auth, tid, [contract]$
6. $C \leftarrow M, \eta_6 : auth, tid, [contract]$

with $contract \triangleq$
$(price, tid, date, [can{:}A], [desc{:}M])$.

By instantiating the exchange modes η_j in the previous scheme, one may generate the *AnBx* variants of the different protocols in the *iKP* family, achieving different security guarantees: this is exactly what we do in Table 7. Notice that all the considered protocols rely on blind forwarding at step 4 to communicate sensitive payment information from the Customer to the Acquirer, without disclosing them to the Merchant. Moreover, a forwarding operation is employed at step 6 to preserve the authenticity of the response by the Acquirer.

5.2. Main Results of iKP Security Verification

We verified the *AnBx* protocols described above and carried out a corresponding analysis of the original specifications of $\{1,2,3\}KP$, as amended in [44]. Below we refer to this amended version as the "original" *iKP*, to be contrasted with the "revised" *AnBx* version in Table 7. In both cases, we ran our tests assuming that the Acquirer is trusted, i.e., encoded as a concrete agent a rather than as a role A; this is often a reasonable assumption in e-payment applications. As we mentioned earlier, the *AnBx* specifications are not just more scalable and remarkably simpler, but they also provide stronger security guarantees, which are detailed in Table 8 and commented further below.

During the analysis of the original $2KP$ and $3KP$ we found a (to the best of our knowledge) new flaw. It is related to the authenticity of the Authorization response *auth* that is generated by the Acquirer and then sent to the other principals at steps 5 and 6. In particular, the starred goals in Table 8 are met only after changing the protocol by adding the identities of Merchant and Customer inside the signature of the Acquirer in the original specification. In $2KP$, since the Customer is not certified, this can be done with an ephemeral identity derived from the credit card number.

It is worth noting that, after the completion of the revised and the amended original $3KP$, each party has evidence of transaction authorization by the other two parties, since the protocol achieves all the authentication goals that can ideally be satisfied, according to the number of certified principals. Moreover, our revised $3KP$, with respect to the original version, provides the additional guarantee of preserving the secrecy of the authorization response *Auth*.

In contrast, the original $3KP$ protocol, the strongest proposed version, fails in two authentication goals: A can only weakly authenticate M and C on $[contract]$. Luckily, if the transaction ID *tid* is unique, this is only a minor problem, since $[contract]$ should also be unique, i.e., two different contracts cannot be confused.

6. SET Purchase Protocol

Secure Electronic Transaction (*SET*) is a family of protocols for securing credit card transactions over insecure networks. This standard was proposed by a consortium of credit card companies and software corporations led by Visa and MasterCard and involving companies like IBM, Microsoft, Netscape, RSA and Verisign. In the present paper we consider the *SET* purchase protocol as outlined in [32]. In the following we distinguish a *signed* and an

mode/step	$\rightarrow$	1KP	2KP	3KP
η_1	$C \rightarrow M$	$(-\|-\|-)$	$(-\|-\|M)$	$@(C\|M\|M)$
η_2	$C \leftarrow M$	$(-\|-\|-)$	$@(M\|C\|-)$	$@(M\|C\|C)$
η_3	$C \rightarrow M$	$(-\|-\|A)$	$(-\|-\|A)$	$(C\|A\|A)$
η_{4a}	$M \rightarrow A$	$(-\|-\|A)$	$(-\|-\|A)$	$(C\|A\|A)$
η_{4b}	$M \rightarrow A$	$(-\|-\|A)$	$@(M\|A\|A)$	$@(M\|A\|A)$
η_5	$M \leftarrow A$	$@(A\|C,M\|-)$	$@(A\|C,M\|M)$	$@(A\|C,M\|M)$
η_6	$C \leftarrow M$	$(A\|C,M\|-)$	$(A\|C,M\|-)$	$(A\|C,M\|C)$
certified agents		A	M,A	C,M,A

Table 7: Exchange modes for the revised *iKP* e-payment protocol

	1KP		2KP		3KP	
Goal	O	R	O	R	O	R
can secret between *C,A*	+	+	+	+	+	+
A weakly authenticates *C* on *can*	-	-	-	-	+	+
desc secret between *C,M*	+	+	+	+	+	+
auth secret between *C,M,A*	-	-	-	-	-	+
price secret between *C,M,A*	-	-	-	-	-	-
M authenticates *A* on *auth*	+*	+	+*	+	+*	+
C authenticates *A* on *auth*	+	+	+	+	+	+
A authenticates *C* on [*contract*]	-	-	-	-	w	+
M authenticates *C* on [*contract*]	-	-	-	-	+	+
A authenticates *M* on [*contract*]	-	-	+	+	w	+
C authenticates *M* on [*contract*]	-	-	+	+	+	+
C authenticates *A* on [*contract*],*auth*	+	+	+	+	+	+
M authenticates *A* on [*contract*],*auth*	+*	+	+*	+	+*	+
verification time	9h10m	2h08m	12h39m	3h57m	59h36m	4h02m
execution time	1.17s	1.16s	1.23s	1.22s	1.16s	1.28s

* goal satisfied only after fixing the definition of Sig_A [29]
w = only weak authentication

Table 8: Performance and security goals satisfied by Original and Revised *iKP*

unsigned version of *SET*: in the former all the parties possess certified key-pairs, while in the latter the Customer does not.

6.1. Protocol Narration

Given the complexity of *SET*, to ease the comparison with other works on such protocol, in this presentation the information exchanged by the principals is denoted with the names commonly used in *SET* specifications. We introduce some basic concepts of the protocol by simply providing a mapping of the exchanged data to the corresponding information in the bare-bone specification presented in Section 4: this should clarify the role of most of the elements. We can identify *PurchAmt* with *price*, *OrderDesc* with *desc*, *pan* with *can* and *AuthCode* with *auth*. The initial knowledge of the three parties can then be summarized as follows: *C* knows *PurchAmt*, *OrderDesc* and *pan*; *M* knows *PurchAmt* and *OrderDesc*; *A* knows *pan*.

During the protocol run, the principals gen-

erate some identifiers: *LIDM* is a local transaction identifier that the Customer sends to the Merchant, while the Merchant generates another session identifier *XID*; we denote the pair (*LIDM*,*XID*) with *TID*. Finally, we complete our abstraction by stipulating $OIdata = OrderDesc$ and $PIdata = pan$; we let $HOD = ([OIdata{:}M], [PIdata{:}A])$. The latter contains the evidence (digest) of the credit card that the Customer intends to use, and the evidence of the order description that will later be forwarded to the Acquirer. In our model *HOD* plays the role of the *dual signature*, a cryptographic mechanism central to *SET*, which is employed to let the Merchant and the Acquirer agree on the transaction without giving any of them full view of the details. Namely, as we said, the Merchant does not need the customer's credit card number to process an order, but he only needs to know that the payment has been approved by the Acquirer. Conversely, the Acquirer does not need to be aware of the details of the order, but he just needs evidence that a particular payment must be processed.

Although many papers on *SET* [32, 45, 42] focus their attention on the signed version of the protocol, again we note that both versions expose a common pattern which allows for an easy specification in *AnBx*. The narration depicting the common structure of the protocols is reported below:

1. $C \to M, \eta_1 :$ *LIDM*
2. $M \to C, \eta_2 :$ *XID*
3. $C \to M$ (decomposed in two steps to specify different communication modes)
   (a) $C \to M, \eta_{3a} :$ *TID,HOD*
   (b) $C \to M, \eta_{3b} :$ *TID,PurchAmt,HOD,PIdata*
4. $M \to A$ (decomposed in two steps to specify different communication modes)
   (a) $M \to A, \eta_{4a} :$ *TID,PurchAmt,HOD,PIdata*
   (b) $M \to A, \eta_{4b} :$ *TID,PurchAmt,HOD*
5. $A \to M, \eta_5 :$ *TID,HOD,AuthCode*
6. $M \to C, \eta_6 :$ *TID,HOD,AuthCode*

Table 9 shows the communication modes we specify to instantiate the previous protocol template to our revised variants of the unsigned and signed versions of *SET*.

6.2. Main Results of SET Security Verification

We verified the *AnBx* specifications of the *SET* purchase protocol and carried out a corresponding analysis of the original specifications, as reported in [32]. In general, our versions of the protocols satisfy stronger security guarantees than the original ones [32], as reported in Table 10. It is worth noting, in particular, that our revised versions do not suffer from two known flaws affecting the original *SET* specification.

The first flaw [32] involves the fifth step of the protocol, where it is not possible to univocally link the identity of the Acquirer and the Merchant with the on-going transaction and the authorization code. Namely, the original message should be amended to include the identity of the merchant M, otherwise the goal "C authenticates M on *AuthCode*" cannot be satisfied. In our revised version the exchange at step 5 is automatically compiled into a message including the identity of both the Merchant and the Customer, so the problem is solved.

The same implementation also prevents the second flaw, presented in [45]. In that paper the specification of the protocol is more detailed than in [32], as it introduces an additional field *AuthRRTags*, which includes the identity of the Merchant. We tested with OFMC the version of *SET* presented in [45] and verified the presence of the flaw, namely an attack against the purchase phase, which exploits a lack of verification in the payment authorization process. It may allow a dishonest Customer to cheat on an honest Merchant when collaborating with another dishonest Merchant. The attack is based on the fact that neither *LIDM* nor *XID* can be considered unique, so they cannot be used to identify a specific Merchant. Therefore the customer can start a parallel purchase with an accomplice, playing the role of another merchant, and make the Acquirer authorize the payment in favor of the accomplice. Here, again the goal "C authenticates M on *AuthCode*" fails.

During our analysis we also verified that both the original specifications [32, 45] fail to verify the goals "C authenticates A on *AuthCode*" and "C authenticates M on *contract*,*AuthCode*". To overcome this problem the protocol must be fixed in the sixth (and final) step, as already outlined in [42]. This issue also leads us to more interesting considerations on how to prove the authorization of the transaction.

mode/step	$\rightarrow$	*unsigned* SET	*signed* SET
η_1	$C \rightarrow M$	$(-\mid -\mid M)$	$@(C \mid M \mid M)$
η_2	$C \leftarrow M$	$@(M \mid C \mid -)$	$@(M \mid C \mid C)$
η_{3a}	$C \rightarrow M$	$(-\mid -\mid M)$	$@(C \mid M \mid M)$
η_{3b}	$C \rightarrow M$	$(-\mid -\mid A)$	$(C \mid A \mid A)$
η_{4a}	$M \rightarrow A$	$(-\mid -\mid A)$	$(C \mid A \mid A)$
η_{4b}	$M \rightarrow A$	$@(M \mid A \mid A)$	$@(M \mid A \mid A)$
η_5	$M \leftarrow A$	$@(A \mid C, M \mid M)$	$@(A \mid C, M \mid M)$
η_6	$C \leftarrow M$	$@(A \mid C, M \mid -)$	$@(A \mid C, M \mid C)$
certified agents		M, A	C, M, A

Table 9: Exchange modes for the revised *SET* e-payment protocol

	unsigned SET		*signed* SET	
Goal	O	R	O	R
pan secret between *C,A*	+	+	+	+
A weakly authenticates *C* on *pan*	-	-	+	+
OrderDesc secret between *C,M*	+	+	+	+
PurchAmt secret between *C,M,A*	-	-	+	+
AuthCode secret between *C,M,A*	-	-	-	+
M authenticates *A* on *AuthCode*	+	+	+	+
C authenticates *A* on *AuthCode*	-	+	-	+
C authenticates *M* on *AuthCode*	+*	+	+*	+
A authenticates *C* on *contract*	-	-	w	w
M authenticates *C* on *contract*	-	-	+	+
A authenticates *M* on *contract*	-	+	-	+
C authenticates *M* on *contract*	+	+	+	+
C authenticates *A* on *contract,AuthCode*	-	+	-	+
M authenticates *A* on *contract,AuthCode*	+	+	+	+
verification time	2h28m	2h22m	2h01m	2h05m
execution time	1.20s	1.19s	1.10s	1.19s

* goal satisfied only after fixing step 5 as in [32]

w = only weak authentication

for revised *SET*: *contract* = *PriceAmt,TID,*[*PIData:A*],[*OIData:M*]

for original *SET*: *contract* = *PriceAmt,TID,*$\mathsf{hash}(PIData)$,$\mathsf{hash}(OIData)$

Table 10: Performance and security goals satisfied by Original and Revised SET purchase protocol

Proving authorization of the transaction. The previous problem arises from the fact that the Customer does not have any evidence of the origin of *AuthCode* by the Acquirer and she instead has to rely only on information provided by the Merchant. For example, giving to the Customer a proof that the Acquirer authorized the payment requires substantial modification of the sixth step of the protocol. In fact, instead of letting the Merchant sign a message for the Customer, we exploit the *AnBx* forward mode to bring to the Customer the authorization of the

payment signed directly by the Acquirer. It is worth noticing that, employing a *fresh* forward mode in the sixth step, we can achieve the desired strong authenticity goal on the pair, even though the transaction identifier is not unique.

We can then confirm the results outlined in [42], showing that, while *iKP* meets all the non-repudiation goals, the original specification of *SET* does not. It is important to notice that, to achieve non-repudiation, each participant must have sufficient proofs to convince an external verifier that the transaction was actually carried out as she claims. A way to obtain this is to assume that the authentication is obtained by means of digital signatures computed with keys which are valid within a Public Key Infrastructure and are issued by a trusted third party (Certification Authority). Although this limits the way authentic channels in *AnBx* could be implemented, in practice it does not represent a significant restriction, since in the considered protocols digital signatures are the standard means meant to achieve authentication.

7. Conclusions

We presented *AnBx*, the currently most expressive *Alice & Bob*-style language. The distinguishing key-feature of the language is a small, yet powerful, extension of the popular channel abstraction to support message forwarding, which is critical for designing and reasoning about complex protocols involving three or more parties. We analysed the formal details related to the definition of the language, and we proved a semantic equivalence between the ideal behaviour of our channels and a simple cryptographic implementation.

Considering alternative implementations of our channel abstractions (i.e., different CCMs) is certainly possible and worth exploring in the future, but it would require us to adapt the proof of our equivalence results. For instance, it seems that using a challenge-response mechanism rather than sequence numbers to achieve freshness would make the equivalence proof quite harder.

We have demonstrated the usefulness of the language in two case studies from the e-payment area, namely *iKP* and *SET*, and we argue that the abstraction from low-level security mechanisms turns out to be helpful for protocols designers. Our compiler from *AnBx* to IF is available online[8] along with the related documentation and the source code of our case studies.

Acknowledgements

This work was partially supported by the MIUR Projects SOFT (*Security Oriented Formal Techniques*), IPODS (*Interacting Processes in Open-ended Distributed Systems*) and CINA (*Compositionality, Interaction, Negotiation and Autonomicity*) and by EU FP7 Projects no.216471, AVANTSSAR (*Automated Validation of Trust and Security of Service-oriented Architectures*) and no.318424, FutureID: Shaping the Future of Electronic Identity.

The authors would like to thank Luca Viganò, David Basin, Benedikt Schmidt, Thomas Groß and the anonymous reviewers for their helpful comments.

References

[1] G. Lowe, Casper: a compiler for the analysis of security protocols, Journal of Computer Security 6 (1) (1998) 53–84.

[2] G. Denker, J. Millen, H. Rueß, The CAPSL integrated protocol environment, Tech. Rep. SRI-CSL-2000-02, SRI International, Menlo Park, CA (2000).

[3] F. Jacquemard, M. Rusinowitch, L. Vigneron, Compiling and verifying security protocols, in: LPAR'00, LNCS 1955, 2000, pp. 131–160.

[4] S. Mödersheim, Algebraic Properties in Alice and Bob Notation, in: ARES'09, 2009, pp. 433–440.

[5] Y. Chevalier, M. Rusinowitch, Compiling and securing cryptographic protocols, Information Processing Letters 110 (3) (2010) 116 – 122.

[6] O. Almousa, S. Mödersheim, L. Viganò, Alice and Bob: Reconciling formal models and implementation, in: Programming Languages with Applications to Biology and Security, Vol. 9465 of LNCS, 2015, pp. 66–85.

[8] `http://www.dais.unive.it/~modesti/anbx/`

[7] U. Carlsen, Generating formal cryptographic protocol specifications, in: IEEE S&P'94, 1994, pp. 137–146.

[8] J. Millen, F. Muller, Cryptographic protocol generation from CAPSL, Tech. Rep. SRI-CSL-01-07, SRI International (2001).

[9] M. Jakobsson, K. Sako, R. Impagliazzo, Designated verifier proofs and their applications, in: EUROCRYPT'96, 1996, pp. 143–158.

[10] J. N. Quaresma, C. W. Probst, Protocol implementation generator, in: NordSec'10, 2010.

[11] P. Modesti, Efficient Java code generation of security protocols specified in An-B/AnBx, in: STM'14, 2014, pp. 204–208.

[12] P. Modesti, Anbx: Automatic generation and verification of security protocols implementations, in: FPS'15, Vol. 9482 of LNCS, Springer, 2016.

[13] M. Abadi, C. Fournet, G. Gonthier, Authentication primitives and their compilation, in: POPL'00, ACM New York, NY, USA, 2000, pp. 302–315.

[14] P. Adao, C. Fournet, Cryptographically sound implementations for communicating processes, in: ICALP'06, Vol. 4052, Springer, 2006, pp. 83–94.

[15] R. Corin, P.-M. Dénielou, C. Fournet, K. Bhargavan, J. J. Leifer, Secure implementations of typed session abstractions, in: CSF'07, IEEE, 2007, pp. 170–186.

[16] K. Bhargavan, R. Corin, P.-M. Dénielou, C. Fournet, J. J. Leifer, Cryptographic protocol synthesis and verification for multiparty sessions, in: CSF'09, 2009.

[17] C. Dilloway, G. Lowe, On the specification of secure channels, in: WITS'07, 2007.

[18] M. Bugliesi, R. Focardi, Language based secure communication, in: CSF'08, 2008, pp. 3–16.

[19] A. Armando, R. Carbone, L. Compagna, LTL model checking for security protocols, in: CSF'07, 2007, pp. 385–396.

[20] A. Kamil, G. Lowe, Specifying and modelling secure channels in strand spaces, in: FAST'09, 2009.

[21] A. Kamil, G. Lowe, Understanding abstractions of secure channels, in: FAST'11, Springer, 2011, pp. 50–64.

[22] S. Mödersheim, L. Viganò, Secure pseudonymous channels, in: M. Backes, P. Ning (Eds.), ESORICS'09, Vol. 5789 of LNCS, Springer, 2009, pp. 337–354.

[23] S. Mödersheim, L. Viganò, Sufficient conditions for vertical composition of security protocols, in: ASIA CCS '14, ACM, 2014, pp. 435–446.

[24] T. Gibson-Robinson, Analysing layered security protocols, Ph.D. thesis, University of Oxford (2013).

[25] C. Sprenger, D. Basin, Developing security protocols by refinement, in: CCS'10, ACM Press, 2010.

[26] C. Sprenger, D. Basin, Refining key establishment, in: CSF'12, Vol. 0, IEEE Computer Society, Los Alamitos, CA, USA, 2012, pp. 230–246.

[27] D. Basin, S. Mödersheim, L. Viganò, OFMC: A symbolic model checker for security protocols, International Journal of Information Security 4 (3) (2005) 181–208.

[28] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, M. Waidner, iKP a family of secure electronic payment protocols, in: 1st USENIX Workshop on Electronic Commerce, 1995.

[29] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreweghen, M. Waidner, Design, implementation, and deployment of the iKP secure electronic payment system, IEEE Journal on Selected Areas in Communications 18 (4) (2000) 611–627.

[30] G. Bella, F. Massacci, L. Paulson, Verifying the SET registration protocols, IEEE Journal on Selected Areas in Communications 21 (1) (2003) 77–87.

[31] G. Bella, F. Massacci, L. Paulson, An overview of the verification of SET, International Journal of Information Security 4 (1) (2005) 17–28.

[32] G. Bella, F. Massacci, L. Paulson, Verifying the SET purchase protocols, Journal of Automated Reasoning 36 (1) (2006) 5–37.

[33] M. Bugliesi, P. Modesti, AnBx - Security protocols design and verification, in: ARSPA-WITS'10, Springer-Verlag, 2010, pp. 164–184.

[34] G. Lowe, A hierarchy of authentication specifications, in: CSFW'97, IEEE Computer Society Press, 1997, pp. 31–43.

[35] AVISPA, Deliverable 2.3: The Intermediate Format, available at `www.avispa-project.org` (2003).

[36] S. Mödersheim, Algebraic properties in Alice and Bob notation (extended version), Tech. Rep. RZ3709, IBM Zurich Research Lab, `domino.research.ibm.com/library/cyberdig.nsf` (2008).

[37] J. Heather, G. Lowe, S. Schneider, How to prevent type flaw attacks on security protocols, Journal of Computer Security 11 (2) (2003) 217–244.

[38] M. Arapinis, M. Duflot, Bounding messages for free in security protocols, in: V. Arvind, S. Prasad (Eds.), FSTTCS 2007, Vol. 4855 of LNCS, Springer, New Delhi, India, 2007, pp. 376–387.

[39] S. Mödersheim, Deciding security for a fragment of ASLan, in: ESORICS'12, 2012, pp. 127–144.

[40] B. Blanchet, An efficient cryptographic protocol verifier based on Prolog rules, in: CSFW'14, IEEE Computer Society, Cape Breton, Nova Scotia, Canada, 2001, pp. 82–96.

[41] A. Armando, L. Compagna, SAT-based Model-Checking for Security Protocols Analysis, International Journal of Information Security 6 (1) (2007) 3–32.

[42] E. Van Herreweghen, Non-repudiation in SET: Open issues, in: FC'01, LNCS, Springer, 2001, pp. 140–156.

[43] D. O'Mahony, M. Peirce, H. Tewari, Electronic payment systems for e-commerce, Artech House Publishers, 2001.

[44] K. Ogata, K. Futatsugi, Formal analysis of the iKP electronic payment protocols, in: ISSS'02, LNCS, Springer, 2003, pp. 441–460.

[45] S. Brlek, S. Hamadou, J. Mullins, A flaw in the electronic commerce protocol SET, Information Processing Letters 97 (3) (2006) 104–108.

[46] J. K. Millen, V. Shmatikov, Constraint solving for bounded-process cryptographic protocol analysis, in: CCS'01, ACM Press, 2001, pp. 166–175.

[47] M. Rusinowitch, M. Turuani, Protocol insecurity with a finite number of sessions, composed keys is NP-complete, Theor. Comput. Sci. 1-3 (299) (2003) 451–475.

[48] D. Basin, S. Mödersheim, L. Viganò, OFMC: A symbolic model checker for security protocols, International Journal of Information Security 4 (3) (2005) 181–208.

Appendix A. Proof of Theorem 2

The idea behind the proof is to abuse a popular verification technique as a proof argument, namely the symbolic constraint-based approach that we call "the lazy intruder" [46, 47, 48, 39]. The intuition behind the lazy intruder is as follows. Every trace can be seen as an instance of a *symbolic trace*, i.e., a sequence of transition rule applications where we delay the unification of left-hand side $\mathsf{ik}(m)$ facts and leave variables in there uninstantiated. Instead, we keep a *constraint* $M \vdash m$, where M is the set of messages the intruder knows at that state. Such a constraint expresses that the intruder must be able to generate the message m from knowledge M. Thus, these constraints before reduction contain only messages m, or instances thereof, for which $\mathsf{ik}(m)$ occurs in the IF specification of the protocol P (in the transition rules of the honest agents). It can be shown that, if there is an attack trace, then there is a corresponding symbolic trace with satisfiable intruder constraints, hence in the proof we can focus without loss of generality on such symbolic traces.

The lazy intruder technique is based on a calculus of constraint reduction rules for checking their satisfiability (and, if satisfiable, determine a solution). There are three constraint reduction rules: GENERATE (to compose new messages from public function symbols), ANALYZE (to obtain all subterms of known messages by decryption and projection) and UNIFY, which states that a possible solution to the constraint $M \vdash t$ exists if there is a $s \in M$, both s and t are not variables, s and t have the most general unifier σ, and all other constraints are satisfiable under σ. The formal constraint reduction rules are reported below, where we let ϕ range over $M \vdash t$ constraints or conjunctions thereof. For the ANALYZE rule we give only the example of asymmetric decryption, other rules are similar.

$$\textsc{Unify}\ (\sigma \in mgu(s,t) \text{ and } s,t \notin \mathcal{V}) \quad \frac{\phi\sigma}{\phi \wedge (M \cup \{s\} \vdash t)}$$

$$\textsc{Generate}\ (f \text{ public}) \quad \frac{\phi \wedge (M \vdash t_1) \wedge \ldots \wedge (M \vdash t_n)}{\phi \wedge (M \vdash f(t_1,\ldots,t_n))}$$

$$\textsc{Analyze}\ (\{m\}_k \in M) \quad \frac{\phi \wedge (M \vdash \mathsf{inv}(k)) \wedge (M \cup \{m\} \vdash t)}{\phi \wedge (M \vdash t)}$$

We can finally prove the theorem.

Restatement of Theorem 2. *If there is an attack against a typeable protocol, then there is a well-typed one, i.e., where every variable X is instantiated with a term t such that $\Gamma(X) = \Gamma(t)$.*

Proof. Consider an arbitrary attack trace and consider its corresponding symbolic trace. By the completeness of the constraint reduction, we know that the constraint reduction will find a solution (i.e., a substitution solving all constraints). We show that any such solution is well-typed. Hence, the existence of an attack implies the existence of a well-typed one.

Technically, we actually need to prove a stronger result by induction over the entire constraint reduction: we prove that every message occurring in the constraints, and any arbitrary subterm of it, is either a variable or an instance of a message in $SMP(P)$, and that all variables are only instantiated in a well-typed way.

Let us first consider a protocol P such that no element of $MP(P)$ is a variable, i.e., P does not involve any step where a "bare value" is transmitted, but all messages are composed terms or constants. In this case, $MP(P) \subseteq SMP(P)$, i.e., the union of the initial intruder knowledge and the messages exchanged in P is included in $SMP(P)$. The GENERATE and ANALYZE cases are straightforward to handle, since, in particular, such rules do not instantiate any variable. In the UNIFY case, both s and t must be well-typed instances of elements of SMP by induction hypothesis, since they are not variables. Given that s and t have a unifier, the typeability assumption implies $\Gamma(s) = \Gamma(t)$, hence also all corresponding subterms of s and t must have the same type by definition of Γ, and the substitution σ is hence well-typed.

Finally, we extend the proof to any protocol P we excluded above, i.e., such that there exists a variable in $MP(P)$. Let P' be a modification of P where every "bare variable" X is replaced by the composed term (t,X) for some fresh tag t that is known to the intruder. Assume now that P has an attack, then also P' has an attack, since the previous wrapping does not enforce any protection. By construction P' satisfies the hypotheses of the previous point, hence for any attack on P' there is a well-typed attack on P', but it is immediate that such well-typed attack works also on P when removing the tag t. □